NAME

cryptsetup-luksFormat − initialize a LUKS partition and set the initial passphrase

SYNOPSIS

cryptsetup *luksFormat* **[<options>] <device> [<key file>]**

DESCRIPTION

Initializes a LUKS partition and sets the passphrase via prompting or <key file>. Note that if the second argument is present, the passphrase is taken from the file given there, without using the −−key−file option. Also note that for both forms of reading the passphrase from a file, you can give '−' as a file name, which results in the passphrase being read from stdin and the safety question being skipped.

You cannot call luksFormat on a device or filesystem that is mapped or in use, e.g., a mounted filesystem, used in LVM, active RAID member, etc. The device or filesystem has to be unmounted in order to call luksFormat.

To enforce a specific version of LUKS format, use −−*type luks1* or *type luks2*. The default format is LUKS2.

To use hardware encryption on an OPAL self−encrypting drive, use −−hw−opal or −−hw−opal−only. Note that some OPAL drives can require a PSID reset (with deletion of data) before using the LUKS format with OPAL options. See −−hw−opal−factory−reset option in cryptsetup *erase* command.

Doing a luksFormat on an existing LUKS container will regenerate the volume key. Unless you have a header backup, all old encrypted data in the container will be permanently irretrievable. Note that luksFormat does not wipe or overwrite the data area. It only creates a new LUKS header with fresh keyslots. See cryptsetup FAQ for more info on how to wipe the whole device, including encrypted data.

<options> can be [−−hash, −−cipher, −−verify−passphrase, −−key−size, −−key−slot, −−key−file (takes precedence over optional second argument), −−keyfile−offset, −−keyfile−size, −−use−random, −−use−urandom, −−uuid, −−volume−key−file, −−iter−time, −−header, −−pbkdf−force−iterations, −−force−password, −−disable−locks, −−timeout, −−type, −−offset, −−align−payload (DEPRECATED)].

For LUKS2, additional **<options>** can be [−−integrity, −−integrity−no−wipe, −−sector−size, −−label, −−subsystem, −−pbkdf, −−pbkdf−memory, −−pbkdf−parallel, −−disable−locks, −−disable−keyring, −−luks2−metadata−size, −−luks2−keyslots−size, −−keyslot−cipher, −−keyslot−key−size, −−integrity−legacy−padding, −−hw−opal, −−hw−opal−only].

OPTIONS

−−align−payload <*number of 512 byte sectors*> (DEPRECATED, use −−offset)
Align payload at a boundary of *value* 512−byte sectors.

If not specified, cryptsetup tries to use the topology info provided by the kernel for the underlying device to get the optimal alignment. If not available (or the calculated value is a multiple of the default), data is by default aligned to a 1MiB boundary (i.e., 2048 512−byte sectors).

For a detached LUKS header, this option specifies the offset on the data device. See also the −−header option.

This option is DEPRECATED and has an unexpected impact on the data offset and keyslot area size (for LUKS2) due to the complex rounding. For fixed data device offset, use −−offset option instead.

−−batch−mode, **−q**
Suppresses all confirmation questions. Use with care!

If the −−verify−passphrase option is not specified, this option also switches off the passphrase

verification.

−−cipher, **−c** <*cipher−spec*>
Set the cipher specification string.

cryptsetup −−help shows the compiled−in defaults.

If a hash is part of the cipher specification, then it is used as part of the IV generation. For example, ESSIV needs a hash function, while "plain64" does not and hence none is specified.

For XTS mode, you can optionally set a key size of 512 bits with the −s option. Key size for XTS mode is twice that for other modes for the same security level.

−−debug or **−−debug−json**
Run in debug mode with full diagnostic logs. Debug output lines are always prefixed by #.

If −−debug−json is used, additional LUKS2 JSON data structures are printed.

−−disable−blkid
Disable use of the blkid library for checking and wiping on−disk signatures.

−−disable−keyring
Do not load the volume key in the kernel keyring; store it directly in the dm−crypt target instead. This option is supported only for the LUKS2 type.

−−disable−locks
Disable lock protection for metadata on disk. This option is valid only for LUKS2 and is ignored for other formats.

WARNING: Do not use this option unless you run cryptsetup in a restricted environment where locking is impossible to perform (where /run directory cannot be used).

−−force−password
Do not use password quality checking for new LUKS passwords.

This option is ignored if cryptsetup is built without password quality checking support.

For more info about password quality check, see the manual page for **pwquality.conf**(5) and **passwdqc.conf**(5).

−−hash, **−h** <*hash−spec*>
Specifies the hash used in the LUKS key setup scheme and volume key digest. The specified hash is used for PBKDF2 and the AF splitter.

The hash algorithm must provide at least 160 bits of output. Do not use a non−crypto hash like **xxhash** as this breaks security. Use *cryptsetup −−help* to show the defaults.

−−header <*device or file storing the LUKS header*>
Use a detached (separated) metadata device or file where the LUKS header is stored. This option allows one to store the ciphertext and LUKS header on different devices.

With a file name as the argument to −−header, the file will be automatically created if it does not exist. See the cryptsetup FAQ for header size calculation.

The −−align−payload option is taken as absolute sector alignment on the ciphertext device and can be zero.

−−help, **−?**
Show help text and default parameters.

−−hw−opal
Format LUKS2 device with dm−crypt encryption stacked on top of HW−based encryption configured on SED OPAL locking range. This option enables both SW and HW based data encryption.

−−hw−opal−only
Format LUKS2 device with HW based encryption configured on SED OPAL locking range only. LUKS2 format only manages the locking range unlock key. This option enables HW−based data encryption managed by the SED OPAL drive only.

Please note that with OPAL−only (−−hw−opal−only) encryption, the configured OPAL administrator PIN (passphrase) allows unlocking all configured locking ranges without LUKS keyslot decryption (without knowledge of LUKS passphrase). Because of many observed problems with compatibility, cryptsetup currently DOES NOT use OPAL single−user mode, which would allow such decoupling of OPAL admin PIN access.

−−integrity <*integrity algorithm*>
Specify the integrity algorithm to be used for authenticated disk encryption in LUKS2.

WARNING: This extension is EXPERIMENTAL and requires dm−integrity kernel target. For native AEAD modes, also enable "User−space interface for AEAD cipher algorithms" in the "Cryptographic API" section (CONFIG_CRYPTO_USER_API_AEAD .config option).

For more info, see the *AUTHENTICATED DISK ENCRYPTION* section in **cryptsetup**(8).

−−integrity−inline
Store integrity tags in hardware sector integrity fields. The device must support sectors with additional protection information (PI, also known as DIF − data integrity field) of the requested size. Another storage subsystem must not use the additional field (the device must present a "nop" profile in the kernel). Note that some devices must be reformatted at a low level to support this option; for NVMe devices, see nvme(1) id−ns LBA profiles.

No journal or bitmap is used in this mode. The device should operate with native speed (without any overhead). This option is available since the Linux kernel version 6.11.

−−integrity−key−size *bits*
The size of the data integrity key. The argument has to be a multiple of 8. Configurable only for HMAC integrity. The default integrity key size is set to the same as the hash output length.

−−integrity−legacy−padding
Use inefficient legacy padding.

Do not use this option until you need compatibility with a specific old kernel.

−−integrity−no−wipe
Skip wiping of device authentication (integrity) tags. If you skip this step, sectors will report an invalid integrity tag until an application writes to the sector.

Skipping this step could also cause write failures due to IO operation alignments. For example, kernel

page cache can request a read of a full page that fails due to an uninitialized integrity tag. It is usually a bug in the application that tries to read data that was not written before.

−−iter−time, **−i** <*number of milliseconds*>
The number of milliseconds to spend with PBKDF passphrase processing. Specifying 0 as a parameter selects the compiled−in default.

−−key−description *text*
Set the key description in the keyring that will be used for passphrase retrieval.

−−key−file, **−d** *file*
Read the passphrase from the file.

If the name given is "−", then the passphrase will be read from stdin. In this case, reading will not stop at newline characters.

See section *NOTES ON PASSPHRASE PROCESSING* in **cryptsetup**(8) for more information.

−−keyfile−offset *value*
Skip *value* bytes at the beginning of the key file.

−−keyfile−size, **−l** *value*
Read a maximum of *value* bytes from the key file. The default is to read the whole file up to the compiled−in maximum that can be queried with −−help. Supplying more data than the compiled−in maximum aborts the operation.

This option is useful to cut trailing newlines, for example. If −−keyfile−offset is also given, the size count starts after the offset.

−−key−size, **−s** *bits*
Sets key size in *bits*. The argument has to be a multiple of 8. The possible key sizes are limited by the cipher and mode used.

See /proc/crypto for more information. Note that the key size in /proc/crypto is stated in bytes.

This option can be used for *open −−type plain* or *luksFormat*. All other LUKS actions will use the key size specified in the LUKS header. Use *cryptsetup −−help* to show the compiled−in defaults.

−−key−slot, **−S** <*0−N*>
For LUKS operations that add key material, this option allows you to specify which keyslot is selected for the new key.

The maximum number of keyslots depends on the LUKS version. LUKS1 can have up to 8 keyslots. LUKS2 can have up to 32 keyslots based on keyslot area size and key size, but a valid keyslot ID can always be between 0 and 31 for LUKS2.

−−keyslot−cipher <*cipher−spec*>
This option can be used to set specific cipher encryption for the LUKS2 keyslot area.

−−keyslot−key−size <*bits*>
This option can be used to set a specific key size for the LUKS2 keyslot area.

−−label <*label*>, **−−subsystem** <*subsystem*>
Set label and subsystem description for LUKS2 device. These are similar to filesystem labels. The

label and subsystem are optional fields and can be later used in udev scripts to trigger user actions once the device marked by these labels is detected.

−−luks2−keyslots−size *size*
This option can be used to set a specific size of the LUKS2 binary keyslot area (key material is encrypted there). The value must be aligned to a multiple of 4096 bytes with a maximum size 128MB. The <size> can be specified with a unit suffix (for example, 128k).

−−luks2−metadata−size *size*
This option can be used to enlarge the LUKS2 metadata (JSON) area. The size includes 4096 bytes for binary metadata (usable JSON area is smaller of the binary area). According to the LUKS2 specification, only these values are valid: 16, 32, 64, 128, 256, 512, 1024, 2048 and 4096 kB. The <size> can be specified with a unit suffix (for example, 128k).

−−offset, **−o** <*number of 512 byte sectors*>
Start offset in the backend device in 512−byte sectors.

The −−offset option sets the data offset (payload) of the data device and must be aligned to 4096−byte sectors (must be a multiple of 8). This option cannot be combined with −−align−payload option.

−−pbkdf <*PBKDF spec*>
Set Password−Based Key Derivation Function (PBKDF) algorithm for LUKS keyslot. The PBKDF can be: *pbkdf2* (for PBKDF2 according to RFC2898), *argon2i* for Argon2i or *argon2id* for Argon2id (see Argon2 <https://www.cryptolux.org/index.php/Argon2> for more info).

For LUKS1, only PBKDF2 is accepted (no need to use this option). The default PBKDF for LUKS2 is set during compilation time and is available in the *cryptsetup −−help* output.

A PBKDF is used for increasing the dictionary and brute−force attack cost for keyslot passwords. The parameters can be time, memory and parallel cost.

For PBKDF2, only the time cost (number of iterations) applies. For Argon2i/id, there is also memory cost (memory required during the process of key derivation) and parallel cost (number of threads that run in parallel during the key derivation.

Note that increasing memory cost also increases time, so the final parameter values are measured by a benchmark. The benchmark tries to find iteration time (−−iter−time) with required memory cost −−pbkdf−memory. If it is not possible, the memory cost is decreased as well. The parallel cost −−pbkdf−parallel is constant and is checked against available CPU cores.

You can see all PBKDF parameters for a particular LUKS2 keyslot with the **cryptsetup−luksDump**(8) command.

If you do not want to use benchmark and want to specify all parameters directly, use −−pbkdf−force−iterations with −−pbkdf−memory and −−pbkdf−parallel. This will override the values without benchmarking. Note it can cause extremely long unlocking time or cause out−of−memory conditions with unconditional process termination. Use only in specific cases, for example, if you know that the formatted device will be used on some small embedded system.

MINIMAL AND MAXIMAL PBKDF COSTS: For **PBKDF2**, the minimum iteration count is 1000 and the maximum is 4294967295 (maximum for 32−bit unsigned integer). Memory and parallel costs are not supported for PBKDF2. For **Argon2i** and **Argon2id**, the minimum iteration count (CPU cost) is 4, and the maximum is 4294967295 (maximum for a 32−bit unsigned integer). Minimum memory cost is 32 KiB and maximum is 4 GiB. If the memory cost parameter is benchmarked (not specified by

a parameter), it is always in the range from 64 MiB to 1 GiB. Memory cost above 1GiB (up to the 4GiB maximum) can be setup only by the −−pbkdf−memory parameter. The parallel cost minimum is 1 and maximum 4 (if enough CPU cores are available, otherwise it is decreased by the available CPU cores).

WARNING: Increasing PBKDF computational costs above the mentioned limits provides negligible additional security improvement. While elevated costs significantly increase brute−force overhead, they offer negligible protection against dictionary attacks. The marginal cost increase for processing an entire dictionary remains fundamentally insufficient.

The hardcoded PBKDF limits represent engineered trade−offs between cryptographic security and operational usability. LUKS maintains portability and must be used within a reasonable time on resource−constrained systems.

Cryptsetup deliberately restricts maximum memory cost (4 GiB) and parallel cost (4) parameters due to architectural limitations (like embedded and legacy systems).

PBKDF memory cost mandates actual physical RAM allocation with intensive write operations that must remain in physical RAM. Any swap usage results in unacceptable performance degradation. Memory management often overcommits allocations beyond available physical memory, expecting most allocated memory to remain unused. In such situations, as PBKDF always uses all allocated memory, it frequently causes out−of−memory failures that abort cryptsetup operations.

−−pbkdf−force−iterations *number*
Avoid the PBKDF benchmark and set the time cost (iterations) directly. It can be used only for a LUKS/LUKS2 device. See −−pbkdf option for more info.

−−pbkdf−memory *number*
Set the memory cost for PBKDF (for Argon2i/id, the number represents kilobytes). Note that it is the maximal value; PBKDF benchmark or available physical memory can decrease it. This option is not available for PBKDF2.

−−pbkdf−parallel *number*
Set the parallel cost for PBKDF (number of threads, up to 4). Note that it is the maximal value; it is decreased automatically if the CPU online count is lower. This option is not available for PBKDF2.

−−progress−frequency *seconds*
Print a separate line every *seconds* with wipe progress.

−−progress−json
Prints progress data in JSON format, which is suitable mostly for machine processing. It prints a separate line every half second (or based on −−progress−frequency value). The JSON output looks as follows during progress (except it's a compact single line):

```
{
  "device":"/dev/sda",      // backing device or file
  "device_bytes":"8192",    // bytes of I/O so far
  "device_size":"44040192", // total bytes of I/O to go
  "speed":"126877696",      // calculated speed in bytes per second (based on pro
  "eta_ms":"2520012",       // estimated time to finish an operation in millisecc
  "time_ms":"5561235"       // total time spent in IO operation in milliseconds
}
```

Note on numbers in JSON output: Due to JSON parser limitations, all numbers are represented in a

string format due to the need for full 64−bit unsigned integers.

−−sector−size *bytes*
Set encryption sector size for use with *LUKS2* device type. It must be a power of two and in the 512 − 4096 bytes range.

The encryption sector size is set based on the underlying data device if not specified explicitly. For native 4096−byte physical sector devices, it is set to 4096 bytes. For 4096/512e (4096−byte physical sector size with 512−byte sector emulation), it is set to 4096 bytes. For drives reporting only a 512−byte physical sector size, it is set to 512 bytes. If the data device is a regular file (container), it is set to 4096 bytes.

If used together with the −−integrity option and dm−integrity journal, the atomicity of writes is guaranteed in all cases (but it costs write performance − data has to be written twice).

Increasing sector size from 512 to 4096 bytes can provide better performance on most modern storage devices and with some hardware encryption accelerators.

Note that using a sector size larger than the underlying storage device's physical sector size may result in data corruption during unexpected power failures. A power failure during write operations may result in only partial completion of the encryption sector write, leaving encrypted data in an inconsistent state that cannot be properly decrypted.

−−timeout, **−t** *seconds*
The number of seconds to wait before a timeout on passphrase input via terminal. It is relevant every time a passphrase is asked. It has no effect if used in conjunction with −−key−file.

This option is useful when the system should not stall if the user does not input a passphrase, e.g., during boot. The default is a value of 0 seconds, which means to wait forever.

−−type *type*
Specifies required device type, for more info, read the *BASIC ACTIONS* section in **cryptsetup**(8).

−−usage
Show short option help.

−−use−random, **−−use−urandom**
For *luksFormat*, these options define which kernel random number generator will be used to create the volume key (which is a long−term key).

Do not use these options with recent kernels (later than version 5.6). For more details, see **NOTES ON RANDOM NUMBER GENERATORS** in **cryptsetup**(8) and **urandom**(4).

−−uuid *UUID*
Use the provided *UUID* for the *luksFormat* command instead of generating a new one. Changes the existing *UUID* when used with the *luksUUID* command.

The *UUID* must be provided in the standard UUID format, e.g., 12345678−1234−1234−1234−123456789abc.

−−verify−passphrase, **−y**
When interactively asking for a passphrase, ask for it twice and complain if both inputs do not match. Ignored on input from file or stdin.

−−version, **−V**
Show the program version.

−−volume−key−file *file*, −−master−key−file file (OBSOLETE alias)
Use a volume key stored in a file.

WARNING: If you create your own volume key, you need to make sure to do it right. Otherwise, you can end up with a low−entropy or otherwise partially predictable volume key, which will compromise security.

REPORTING BUGS

Report bugs at cryptsetup mailing list <cryptsetup@lists.linux.dev> or in Issues project section <https://gitlab.com/cryptsetup/cryptsetup/−/issues/new>.

Please attach the output of the failed command with −−debug option added.

SEE ALSO

Cryptsetup FAQ <https://gitlab.com/cryptsetup/cryptsetup/wikis/FrequentlyAskedQuestions>

cryptsetup(8), **integritysetup**(8) and **veritysetup**(8)

CRYPTSETUP

Part of cryptsetup project <https://gitlab.com/cryptsetup/cryptsetup/>.